Ml

Chronic Kidney Dises.ipynb ×

Code Python 3

```
[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from IPython.core.pylabtools import figsize
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction import DictVectorizer
import xgboost as xgb
from sklearn.model_selection import cross_val_score
from sklearn_pandas import DataFrameMapper, CategoricalImputer
from sklearn.preprocessing import Imputer
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import FunctionTransformer
from missing_value.missing_values_table import missing_values_table
from missing_value.fill_missing_values import Categorical_Imputer
from metrics.roc_auc import roc_auc
from sklearn.metrics import roc_auc_score, roc_curve
```

```
[ ]: # Data Preparation

# Load Data

pd.set_option('display.max_columns', 30)
df = pd.read_csv('datasets_kidney_disease.csv', header=None,
names=['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad'
```

```
[ ]: df.head(10)
```

```
[ ]: # df info

df.info()
```

Out[3]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu
0	48	80	1.020	1	0	?	normal	notpresent	notpresent	121	36
1	7	50	1.020	4	0	?	normal	notpresent	notpresent	?	18
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent	423	53
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	56
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent	106	26
5	60	90	1.015	3	0	?	?	notpresent	notpresent	74	25
6	68	70	1.010	0	0	?	normal	notpresent	notpresent	100	54
7	24	?	1.015	2	4	normal	abnormal	notpresent	notpresent	410	31
8	52	100	1.015	3	0	normal	abnormal	present	notpresent	138	60
9	53	90	1.020	2	0	abnormal	abnormal	present	notpresent	70	107

	sc	sod	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe	ane	class
0	1.2	?	?	15.4	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	0.8	?	?	11.3	38	6000	?	no	no	no	good	no	no	ckd
2	1.8	?	?	9.6	31	7500	?	no	yes	no	poor	no	yes	ckd
3	3.8	111	2.5	11.2	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	1.4	?	?	11.6	35	7300	4.6	no	no	no	good	no	no	ckd
5	1.1	142	3.2	12.2	39	7800	4.4	yes	yes	no	good	yes	no	ckd
6	24.0	104	4.0	12.4	36	?	?	no	no	no	good	no	no	ckd
7	1.1	?	?	12.4	44	6900	5	no	yes	no	good	yes	no	ckd
8	1.9	?	?	10.8	33	9600	4.0	yes	yes	no	good	no	yes	ckd
9	7.2	114	3.7	9.5	29	12100	3.7	yes	yes	no	poor	no	yes	ckd

Columns explain: * age - age * bp - blood pressure * sg - specific gravity * al - albumin * su - sugar * rbc - red blood cells * pc - pus cell * pcc - pus cell clumps * ba - bacteria * bgr - blood glucose random * bu - blood urea * sc - serum creatinine * sod - sodium * pot - potassium * hemo - hemoglobin * pcv - packed cell volume * wc - white blood cell count * rc - red blood cell count * htn - hypertension * dm - diabetes mellitus * cad - coronary artery disease * appet - appetite * pe - pedal edema * ane - anemia * class - class

```
[ ]: # df info

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
age 400 non-null object
bp 400 non-null object
sg 400 non-null object
al 400 non-null object
su 400 non-null object
rbc 400 non-null object
pc 400 non-null object
pcc 400 non-null object
ba 400 non-null object
bgr 400 non-null object
bu 400 non-null object
sc 400 non-null object
sod 400 non-null object
pot 400 non-null object
hemo 400 non-null object
pcv 400 non-null object
wc 400 non-null object
rc 400 non-null object
htn 400 non-null object
dm 400 non-null object
cad 400 non-null object
appet 400 non-null object
pe 400 non-null object
ane 400 non-null object
class 400 non-null object
```

```
[ ]: # repace ? values
df.replace('?', np.nan, inplace=True)
df.head(10)
```

Out[5]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	\
0	48	80	1.020	1	0	NaN	normal	notpresent	notpresent	121	36	
1	7	50	1.020	4	0	NaN	normal	notpresent	notpresent	NaN	18	
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent	423	53	
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	56	
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent	106	26	
5	60	90	1.015	3	0	NaN	NaN	notpresent	notpresent	74	25	
6	68	70	1.010	0	0	NaN	normal	notpresent	notpresent	100	54	
7	24	NaN	1.015	2	4	normal	abnormal	notpresent	notpresent	410	31	
8	52	100	1.015	3	0	normal	abnormal	present	notpresent	138	60	
9	53	90	1.020	2	0	abnormal	abnormal	present	notpresent	70	107	

	sc	sod	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe	ane	class
0	1.2	NaN	NaN	15.4	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	0.8	NaN	NaN	11.3	38	6000	NaN	no	no	no	good	no	no	ckd
2	1.8	NaN	NaN	9.6	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3.8	111	2.5	11.2	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	1.4	NaN	NaN	11.6	35	7300	4.6	no	no	no	good	no	no	ckd
5	1.1	142	3.2	12.2	39	7800	4.4	yes	yes	no	good	yes	no	ckd
6	24.0	104	4.0	12.4	36	NaN	NaN	no	no	no	good	no	no	ckd
7	1.1	NaN	NaN	12.4	44	6900	5	no	yes	no	good	yes	no	ckd
8	1.9	NaN	NaN	10.8	33	9600	4.0	yes	yes	no	good	no	yes	ckd
9	7.2	114	3.7	9.5	29	12100	3.7	yes	yes	no	poor	no	yes	ckd

[ ]:
```python
# Check missing value percentage
 # missing value table
missing_values_table(df)
```

Out[7]:

	Missing Values	% of Total Values
rbc	152	38.00
rc	131	32.75
wc	106	26.50
pot	88	22.00
sod	87	21.75
pcv	71	17.75
pc	65	16.25
hemo	52	13.00
su	49	12.25
sg	47	11.75
al	46	11.50
bgr	44	11.00
bu	19	4.75
sc	17	4.25
bp	12	3.00
age	9	2.25
ba	4	1.00
pcc	4	1.00
htn	2	0.50
dm	2	0.50
cad	2	0.50
appet	1	0.25
pe	1	0.25
ane	1	0.25

[ ]:
```python
# numerical columns
num_cols = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv
```

[ ]:
```python
# categorical columns
cate_cols = df.columns.drop('class').drop(num_cols)
# display categorical columns
cate_cols
```

Out[9]: Index(['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane'], dtype='objec

[ ]:
```python
# convert numerical data
df[num_cols] = df[num_cols].apply(pd.to_numeric, errors='coerce')
```

[ ]:
```python
# df info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
age      391 non-null float64
bp       388 non-null float64
sg       353 non-null float64
al       354 non-null float64
su       351 non-null float64
rbc      248 non-null object
pc       335 non-null object
pcc      396 non-null object
ba       396 non-null object
bgr      356 non-null float64
bu       381 non-null float64
sc       383 non-null float64
sod      313 non-null float64
pot      312 non-null float64
hemo     348 non-null float64
pcv      329 non-null float64
wc       294 non-null float64
rc       269 non-null float64
htn      398 non-null object
dm       398 non-null object
cad      398 non-null object
appet    399 non-null object
pe       399 non-null object
ane      399 non-null object
class    400 non-null object
dtypes: float64(14), object(11)
memory usage: 78.2+ KB
```

[ ]:

```
# Categorical Feature Unique Values
# categorical columns
cate_cols = df.columns.drop('class').drop(num_cols)
# display categorical columns
cate_cols
```

Out[12]:

```
Index(['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane'], dtype='obj
```

[ ]:

```
# check the number of unique values
df[cate_cols].apply(lambda x: x.nunique(), axis=0)
```

```
Out[13]: rbc      2
         pc       2
         pcc      2
         ba       2
         htn      2
         dm       3
         cad      2
         appet    2
         pe       2
         ane      2
         dtype: int64
```

```python
# Problem found on df['dm'], string has extra space
df['dm'].unique()
```

```python
df['dm'].dtype
```

```
Out[14]: array(['yes', 'no', ' yes', nan], dtype=object)
```

```python
[ ]: # delete the space
     df['dm'] = df['dm'].str.strip()
```

```python
[ ]: # convert categorical target to numerical
     df['class'] = df['class'].apply(lambda x: 1 if x=='ckd' else 0)
```

```python
[ ]: # show the head of df['class']
     df['class'].head()
```

```
Out[17]: 0    1
         1    1
         2    1
         3    1
         4    1
         Name: class, dtype: int64
```

```python
[ ]: # Sklearn Imputer and Pandas Get Dummies Approach
     # load X and y
     X = df.drop(columns=['class'])
     y= df['class']
```

[ ]:
```python
# Imputing Data

# define numerical imputer
num_imputer = Imputer(strategy='median')
```

[ ]:
```python
# imputing on numerical data
X[num_cols] = num_imputer.fit_transform(X[num_cols])
```

[ ]:
```python
# define categorical imputer
cate_imputer = Categorical_Imputer('most_frequent')
```

[ ]:
```python
# imputing on categorical data
X[cate_cols] = cate_imputer.fit_transform(X[cate_cols])
```

[ ]:
```python
X.head()
```

Out[24]:
```
    age    bp     sg   al   su     rbc        pc         pcc          ba  \
0  48.0  80.0  1.020  1.0  0.0  normal    normal  notpresent  notpresent
1   7.0  50.0  1.020  4.0  0.0  normal    normal  notpresent  notpresent
2  62.0  80.0  1.010  2.0  3.0  normal    normal  notpresent  notpresent
3  48.0  70.0  1.005  4.0  0.0  normal  abnormal     present  notpresent
4  51.0  80.0  1.010  2.0  0.0  normal    normal  notpresent  notpresent

     bgr    bu   sc    sod  pot  hemo   pcv      wc   rc  htn   dm cad appet  \
0  121.0  36.0  1.2  138.0  4.4  15.4  44.0  7800.0  5.2  yes  yes  no  good
1  121.0  18.0  0.8  138.0  4.4  11.3  38.0  6000.0  4.8   no   no  no  good
2  423.0  53.0  1.8  138.0  4.4   9.6  31.0  7500.0  4.8   no  yes  no  poor
3  117.0  56.0  3.8  111.0  2.5  11.2  32.0  6700.0  3.9  yes   no  no  poor
4  106.0  26.0  1.4  138.0  4.4  11.6  35.0  7300.0  4.6   no   no  no  good

    pe  ane
0   no   no
1   no   no
2   no  yes
3  yes  yes
4   no   no
```

[ ]:
```python
# missing value table
missing_values_table(X)
```

```
Your slelected dataframe has 24 columns.
There are 0 columns that have missing values.
```

Out[25]:
```
Empty DataFrame
Columns: [Missing Values, % of Total Values]
Index: []
```

```
[ ]: # Get dummies
     X = pd.get_dummies(X, prefix_sep='_', drop_first=True)
     # X head
     X.head()
```

```
Out[26]:   age    bp     sg   al   su    bgr    bu   sc    sod  pot  hemo   pcv  \
        0  48.0  80.0  1.020  1.0  0.0  121.0  36.0  1.2  138.0  4.4  15.4  44.0
        1   7.0  50.0  1.020  4.0  0.0  121.0  18.0  0.8  138.0  4.4  11.3  38.0
        2  62.0  80.0  1.010  2.0  3.0  423.0  53.0  1.8  138.0  4.4   9.6  31.0
        3  48.0  70.0  1.005  4.0  0.0  117.0  56.0  3.8  111.0  2.5  11.2  32.0
        4  51.0  80.0  1.010  2.0  0.0  106.0  26.0  1.4  138.0  4.4  11.6  35.0

               wc   rc  rbc_normal  pc_normal  pcc_present  ba_present  htn_yes  \
        0  7800.0  5.2           1          1            0           0        1
        1  6000.0  4.8           1          1            0           0        0
        2  7500.0  4.8           1          1            0           0        0
        3  6700.0  3.9           1          0            1           0        1
        4  7300.0  4.6           1          1            0           0        0

           dm_yes  cad_yes  appet_poor  pe_yes  ane_yes
        0       1        0           0       0        0
        1       0        0           0       0        0
        2       1        0           1       0        1
        3       0        0           1       1        1
        4       0        0           0       0        0
```

```
# cross validation score

cv_scores = cross_val_score(xgb_cl, X, y, scoring='roc_auc', cv=3)
```

```
# print out the mean cross validation score

print('3-Fold AUC: {}'.format(np.mean(cv_scores)))
```

```
3-Fold AUC: 0.9987177280550775
```

```
[ ]: # Fit Model
      # train test split

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
     random_state=21, stratify=y)
```

```
[ ]: # fit the model

     xgb_cl.fit(X_train, y_train)
```

```
Out[32]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
              max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
              n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=True, subsample=1)
```

```
# ROC AUC Curve
  # predict on the test set
y_pred_prob = xgb_cl.predict_proba(X_test)[:, 1]
```

```
# instantiate a roc_auc object
ROC = roc_auc(y_test, y_pred_prob, model='XGB')
```

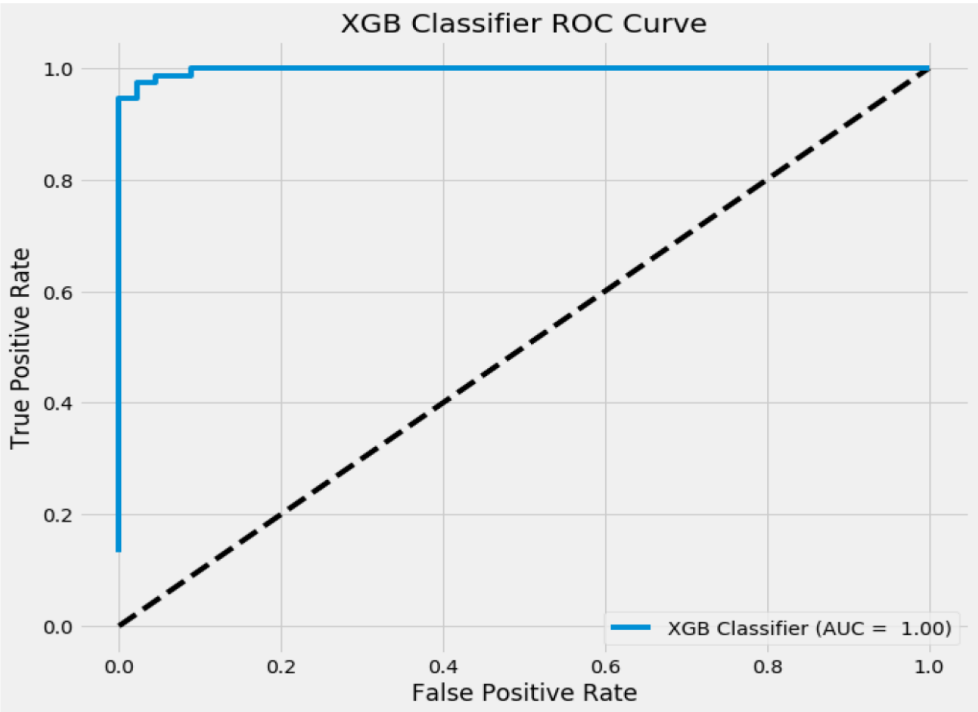
```
# AUC score
ROC.auc()
```

```
Out[35]: 0.9976296296296296
```

```
[ ]: # set figsize
     figsize(10,8)
```

```
[ ]: # plot styple
     plt.style.use('fivethirtyeight')
```

```
[ ]: # plot roc
     ROC.plot_roc()
```

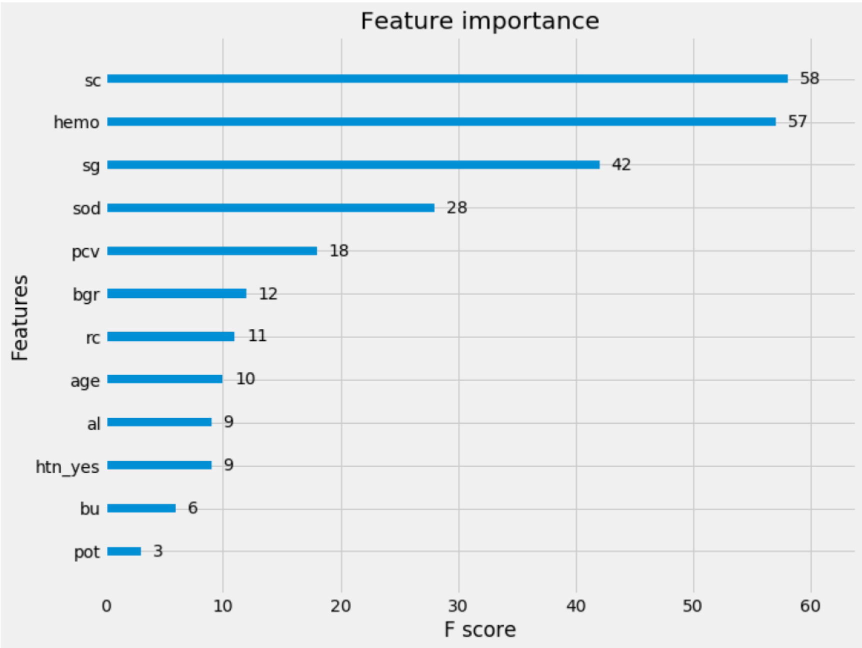


```python
# set figsize
figsize(10,8)
```

```python
# plot styple
plt.style.use('fivethirtyeight')
```

```python
# plot feature importance
xgb.plot_importance(xgb_cl)
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x19539b961d0>




```python
# Imputing: Function Transformer

 # define numerical imputer
num_imputer = Imputer(strategy='median')
```

```python
# define categorical imputer
cate_imputer = Categorical_Imputer('most_frequent')
```

```python
# FunctionTransformer applied on the cate_imputer's fit_transform function
cate_imp = FunctionTransformer(cate_imputer.fit_transform, validate=False)
```

```python
# Numerical Sub-Pipeline

# numerical pipeline
num_pipeline = Pipeline(steps=
[('select_num', select_numeric),
('imp_num', num_imputer)])
```

```python
# Test num_pipeline
num_pipeline.fit_transform(X)
```

```
array([[4.800e+01, 8.000e+01, 1.020e+00, ..., 4.400e+01, 7.800e+03,
         5.200e+00],
        [7.000e+00, 5.000e+01, 1.020e+00, ..., 3.800e+01, 6.000e+03,
         4.800e+00],
        [6.200e+01, 8.000e+01, 1.010e+00, ..., 3.100e+01, 7.500e+03,
         4.800e+00],
        ...,
        [1.200e+01, 8.000e+01, 1.020e+00, ..., 4.900e+01, 6.600e+03,
         5.400e+00],
        [1.700e+01, 6.000e+01, 1.025e+00, ..., 5.100e+01, 7.200e+03,
         5.900e+00],
        [5.800e+01, 8.000e+01, 1.025e+00, ..., 5.300e+01, 6.800e+03,
         6.100e+00]])
```

[ ]:
```
# Categorical Sub-Pipeline
 # categorical pipeline

cate_pipeline = Pipeline(steps=
[('select_cate', select_cate),
('imp_cate', cate_imp)])
```

[ ]:
```
# Test categorical pipeline
cate_pipeline.fit_transform(X).head()
```

Out[47]:

	rbc	pc	pcc	ba	htn	dm	cad	appet	pe	ane
0	normal	normal	notpresent	notpresent	yes	yes	no	good	no	no
1	normal	normal	notpresent	notpresent	no	no	no	good	no	no
2	normal	normal	notpresent	notpresent	no	yes	no	poor	no	yes
3	normal	abnormal	present	notpresent	yes	no	no	poor	yes	yes
4	normal	normal	notpresent	notpresent	no	no	no	good	no	no

If we get dummies after combine numerical and categorical data, FeatureUnion will output a 2D numpy array mixed of numerical and categorical data. Pandas get dummies won't work on numpy array. Even if use pd.DataFrame() turn it into a DataFrame, the mixed numerical and categorical data will cause trouble in dtypes. pd.DataFrame() usually only works on concering a numpy array only containing float to dataframe.

We incorporate get dummies in categorical data sub pipeline.

[ ]:
```
# Make a transformer from a function
get_dummies = FunctionTransformer(lambda x: pd.get_dummies(x, prefix_sep='_',
cate_pipeline = Pipeline(steps=
[('select_cate', select_cate),
('imp_cate', cate_imp),
('get_dummies', get_dummies)])
```

[ ]:
```
# test output
cate_pipeline.fit_transform(X).head()
```

```
Out[50]:    rbc_normal  pc_normal  pcc_present  ba_present  htn_yes  dm_yes  cad_yes  \
         0           1          1            0           0        1       1        0
         1           1          1            0           0        0       0        0
         2           1          1            0           0        0       1        0
         3           1          0            1           0        1       0        0
         4           1          1            0           0        0       0        0

            appet_poor  pe_yes  ane_yes
         0           0       0        0
         1           0       0        0
         2           1       0        1
         3           1       1        1
         4           0       0        0
```

```
[ ]: # Feature Union
      # Combine numerical and categorical processing
     union_impute = FeatureUnion(transformer_list=
     [('numerical', num_pipeline),
     ('categorical', cate_pipeline)])
```

```
[ ]: # Test union_impute
     union_results = union_impute.fit_transform(X)

     # print union_results
     union_results
```

```
Out[52]: array([[48.   , 80.   ,  1.02 , ...,  0.   ,  0.   ,  0.   ],
                [ 7.   , 50.   ,  1.02 , ...,  0.   ,  0.   ,  0.   ],
                [62.   , 80.   ,  1.01 , ...,  1.   ,  0.   ,  1.   ],
                ...,
                [12.   , 80.   ,  1.02 , ...,  0.   ,  0.   ,  0.   ],
                [17.   , 60.   ,  1.025, ...,  0.   ,  0.   ,  0.   ],
                [58.   , 80.   ,  1.025, ...,  0.   ,  0.   ,  0.   ]])
```

FeatureUnion ouputs an array, no longer a DataFrame

```
[ ]: # Classifier
      # XGBClassifier
     xgb_cl = xgb.XGBClassifier()
```

```
[ ]: # Build Pipeline
       # Build pipeline
     pipeline = Pipeline(steps=[
     ('union', union_impute),
     ('classifier', xgb_cl)
     ])
```

```
[ ]: # Cross Validation Score on Pipeline
      cross_val_scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=3)
```

```
[ ]: # print out the mean cross validation score
     print('3-Fold AUC: {}'.format(np.mean(cv_scores)))
```

```
3-Fold AUC: 0.9987177280550775
```

```
# Fit Model
 # train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=21, stratify=y)
```

```
# fit the model
pipeline.fit(X_train, y_train)
```

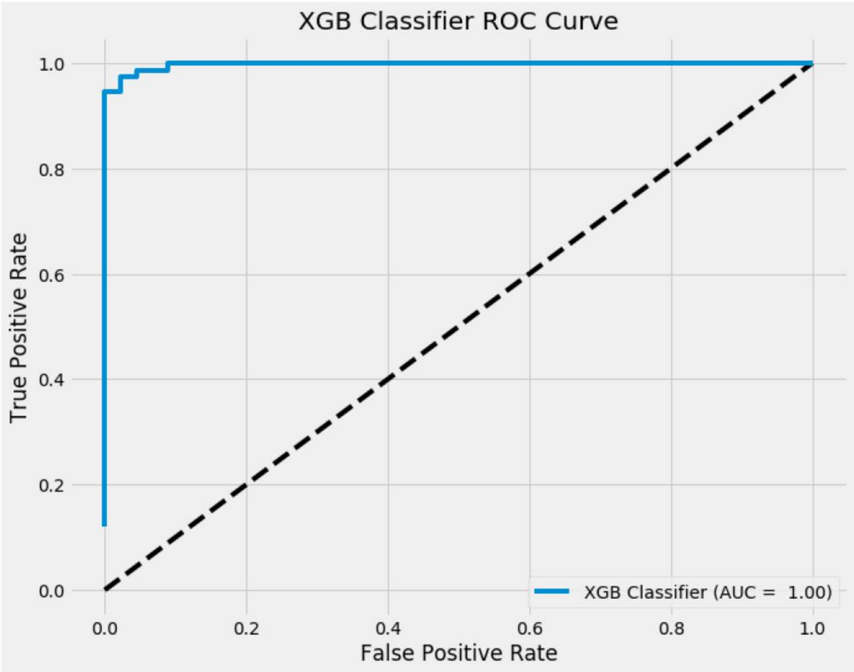
```
Out[59]: Pipeline(memory=None,
              steps=[('union', FeatureUnion(n_jobs=1,
                transformer_list=[('numerical', Pipeline(memory=None,
              steps=[('select_num', FunctionTransformer(accept_sparse=False,
                    func=<function <lambda> at 0x0000019539867378>, inv_kw_args=None,
                    inverse_func=None, kw_args=None, pass_y='depr...
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                silent=True, subsample=1))])
```

```
[ ]: # ROC AUC Curve
      # predict on the test set

     y_pred_prob = pipeline.predict_proba(X_test)[:, 1]
     ROC = roc_auc(y_test, y_pred_prob, model='XGB')
```

```
[ ]:  # AUC score
     ROC.auc()
```

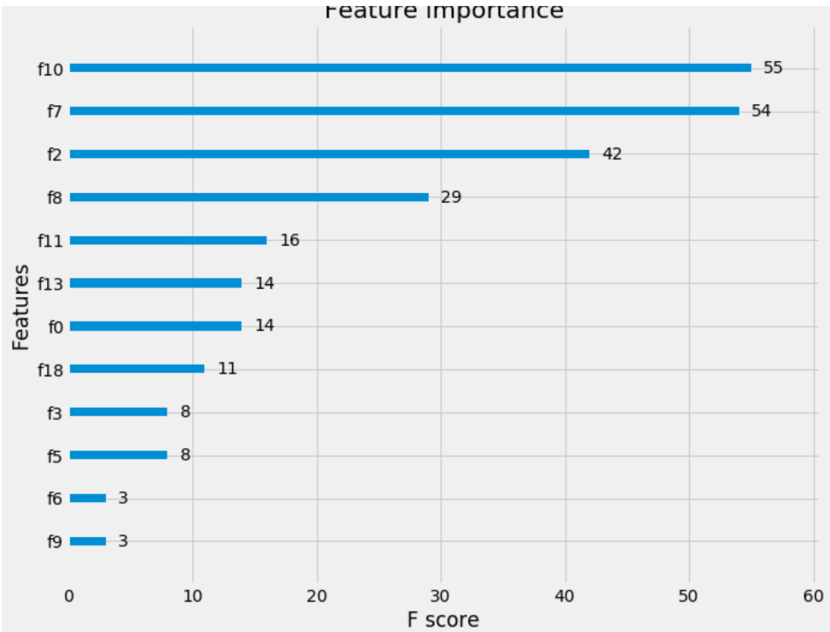
```
Out[62]: 0.9976296296296295
```

```
[ ]: # set figsize
     figsize(10,8)

     # plot styple
     plt.style.use('fivethirtyeight')

     # plot roc
     ROC.plot_roc()
```




```
[ ]: # Feature Importance
      # set figsize

     figsize(10,8)

     # plot styple
     plt.style.use('fivethirtyeight')

     # plot feature importance
     xgb.plot_importance(pipeline.named_steps['classifier'])
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x19539b96048>



[ ]:
```python
# Sklearn_pandas package Approach
# Load X and y

X = df.drop(columns=['class'])
y= df['class']
```

[ ]:
```python
# Creat Boolean Mask
categorical_feature_mask = X.dtypes==object
```

[ ]:
```python
# Categorial feature list & Numerical feature list
 # filter categorical columns using mask and turn it into a list

categorical_cols = X.columns[categorical_feature_mask].tolist()
# show categorical columns
categorical_cols
```

Out[67]:
```
['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane']
```

[ ]:
```python
# filter numerical columns using mask and turn it into a list

numerical_cols = X.columns[~categorical_feature_mask].tolist()

# show numerical columns
numerical_cols
```

Out[68]:
```
['age',
 'bp',
 'sg',
 'al',
 'su',
 'bgr',
 'bu',
 'sc',
 'sod',
 'pot',
 'hemo',
 'pcv',
 'wc',
 'rc']
```

[ ]:
```
# Numerical Imputer: DataFrameMapper
  # Construc numerical imputer
numeric_imputation_mapper = DataFrameMapper(
[([numeric_feature],Imputer(strategy="median")) input_df=True,
df_out=True
)
```

[ ]:
```
 # imputing numerical missing values
X_num = numeric_imputation_mapper.fit_transform(X)
```

[ ]:
```
# Categorical Imputer: DataFrameMapper
  # Apply categorical imputer
categorical_imputation_mapper = DataFrameMapper(
[(category_feature, CategoricalImputer()) input_df=True,
df_out=True
)
```

[ ]:
```
 # imputing categorical missing values
X_cat = categorical_imputation_mapper.fit_transform(X)
```

[ ]:
```
# Concatenate X_num and X_cat
  # concat X
X = pd.concat([X_num, X_cat], axis=1) # axis=1

# show X head
X.head() # X is still a dataframe
```

Out[73]:

	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	\
0	48.0	80.0	1.020	1.0	0.0	121.0	36.0	1.2	138.0	4.4	15.4	44.0	
1	7.0	50.0	1.020	4.0	0.0	121.0	18.0	0.8	138.0	4.4	11.3	38.0	
2	62.0	80.0	1.010	2.0	3.0	423.0	53.0	1.8	138.0	4.4	9.6	31.0	
3	48.0	70.0	1.005	4.0	0.0	117.0	56.0	3.8	111.0	2.5	11.2	32.0	
4	51.0	80.0	1.010	2.0	0.0	106.0	26.0	1.4	138.0	4.4	11.6	35.0	

	wc	rc	rbc	pc	pcc	ba	htn	dm	cad	appet	\
0	7800.0	5.2	normal	normal	notpresent	notpresent	yes	yes	no	good	
1	6000.0	4.8	normal	normal	notpresent	notpresent	no	no	no	good	
2	7500.0	4.8	normal	normal	notpresent	notpresent	no	yes	no	poor	
3	6700.0	3.9	normal	abnormal	present	notpresent	yes	no	no	poor	
4	7300.0	4.6	normal	normal	notpresent	notpresent	no	no	no	good	

	pe	ane
0	no	no
1	no	no
2	no	yes
3	yes	yes
4	no	no

[ ]:
```
# DictVectorizer
 # turn X into dict
X_dict = X.to_dict(orient='records') # turn each row as key-value pairs
# show X_dict
X_dict
```

[ ]:
```python
# instantiate a Dictvectorizer object for X
dv_X = DictVectorizer(sparse=False) # sparse = False makes the output is not a sp
```

[ ]:
```python
# apply dv_X on X_dict
X_encoded = dv_X.fit_transform(X_dict)
# show X_encoded
X_encoded
```

Out[76]:
```
array([[4.80e+01, 1.00e+00, 1.00e+00, ..., 1.38e+02, 0.00e+00, 7.80e+03],
       [7.00e+00, 4.00e+00, 1.00e+00, ..., 1.38e+02, 0.00e+00, 6.00e+03],
       [6.20e+01, 2.00e+00, 0.00e+00, ..., 1.38e+02, 3.00e+00, 7.50e+03],
       ...,
       [1.20e+01, 0.00e+00, 1.00e+00, ..., 1.37e+02, 0.00e+00, 6.60e+03],
       [1.70e+01, 0.00e+00, 1.00e+00, ..., 1.35e+02, 0.00e+00, 7.20e+03],
       [5.80e+01, 0.00e+00, 1.00e+00, ..., 1.41e+02, 0.00e+00, 6.80e+03]])
```

[ ]:
```python
# vocabulary
vocab = dv_X.vocabulary_

# show vocab
vocab
```

Out[77]:
```
{'age': 0,
 'bp': 9,
 'sg': 30,
 'al': 1,
 'su': 32,
 'bgr': 8,
 'bu': 10,
 'sc': 29,
 'sod': 31,
 'pot': 25,
 'hemo': 15,
 'pcv': 22,
 'wc': 33,
 'rc': 28,
 'rbc=normal': 27,
 'pc=normal': 19,
 'pcc=notpresent': 20,
 'ba=notpresent': 6,
 'htn=yes': 17,
 'dm=yes': 14,
 'cad=no': 11,
 'appet=good': 4,
 'pe=no': 23,
 'ane=no': 2,
 'htn=no': 16,
 'dm=no': 13,
 'appet=poor': 5,
 'ane=yes': 3,
 'pc=abnormal': 18,
 'pcc=present': 21,
 'pe=yes': 24,
 'rbc=abnormal': 26,
 'cad=yes': 12,
 'ba=present': 7}
```

[ ]:
```python
# sort vocabulary
sorted_vocab = sorted(vocab.items(), key=lambda x: x[1])

# show sorted_vocab
sorted_vocab
```

```
Out[78]: [('age', 0),
          ('al', 1),
          ('ane=no', 2),
          ('ane=yes', 3),
          ('appet=good', 4),
          ('appet=poor', 5),
          ('ba=notpresent', 6),
          ('ba=present', 7),
          ('bgr', 8),
          ('bp', 9),
          ('bu', 10),
          ('cad=no', 11),
          ('cad=yes', 12),
          ('dm=no', 13),
          ('dm=yes', 14),
          ('hemo', 15),
          ('htn=no', 16),
          ('htn=yes', 17),
          ('pc=abnormal', 18),
          ('pc=normal', 19),
          ('pcc=notpresent', 20),
          ('pcc=present', 21),
          ('pcv', 22),
          ('pe=no', 23),
          ('pe=yes', 24),
          ('pot', 25),
          ('rbc=abnormal', 26),
          ('rbc=normal', 27),
          ('rc', 28),
          ('sc', 29),
          ('sg', 30),
          ('sod', 31),
          ('su', 32),
          ('wc', 33)]
```

```
3-Fold AUC: 0.9987177280550775
```

```
[ ]: # Cross Validation
In [79]: # Instantiate XGBClassifier
xgb_cl = xgb.XGBClassifier()
In [80]: # cross validation score
cv_scores = cross_val_score(xgb_cl, X_encoded, y, scoring='roc_auc', cv=3)
223
In [81]: # print out the mean cross validation score
print('3-Fold AUC: {}'.format(np.mean(cv_scores)))
```

```
[ ]: # Fit Model
 # train test split
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3,
random_state=21, stratify=y)
```

```
[ ]:  # fit the model
xgb_cl.fit(X_train, y_train)
```

```
Out[83]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
              max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
              n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=True, subsample=1)
```

```
[ ]: # Fit Model
      # train test split
     X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3,
     random_state=21, stratify=y)
```

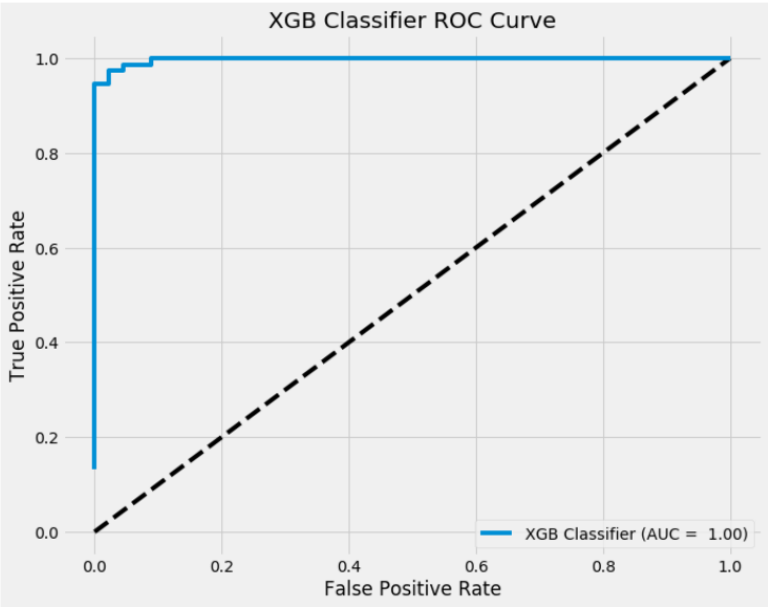
```
[ ]:  # fit the model
     xgb_cl.fit(X_train, y_train)
```

```
[ ]: # ROC AUC Curve
      # predict on the test set
     y_pred_prob = xgb_cl.predict_proba(X_test)[:, 1] # [:, 1]: the second value is
     ROC = roc_auc(y_test, y_pred_prob, model='XGB')
```

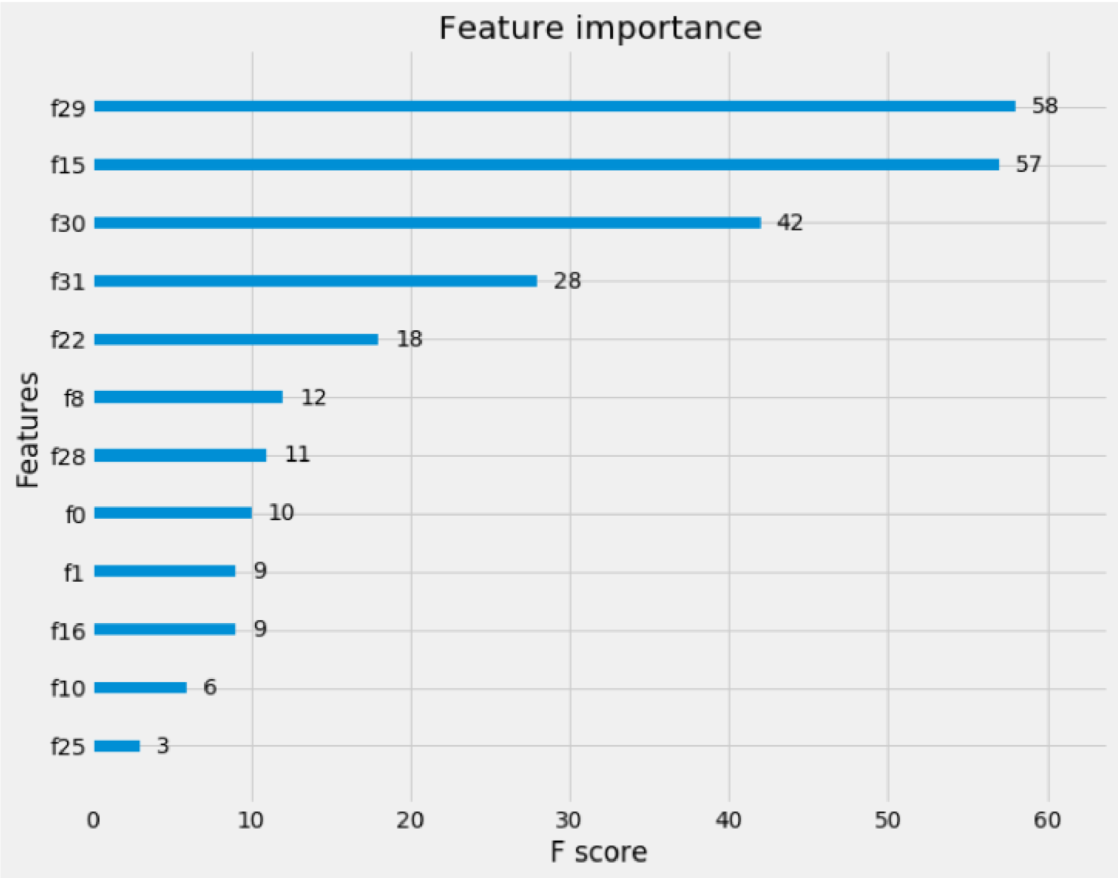
```
[ ]:  # AUC score
     ROC.auc()
```

```
Out[86]: 0.9976296296296296
```

```
# set figsize
figsize(10,8)

# plot styple
plt.style.use('fivethirtyeight')

# plot roc
ROC.plot_roc()
```



```
[ ]: #Feature Importance
      # set figsize

     figsize(10,8)

     # plot styple

     plt.style.use('fivethirtyeight')

     # plot feature importance

     xgb.plot_importance(xgb_cl)
```



[ ]:

```
# Check number of nulls in each feature column

nulls_per_column = X.isnull().sum()
print(nulls_per_column)

# Create a boolean mask for categorical columns

categorical_feature_mask = X.dtypes == object

# Get list of categorical column names

categorical_columns = X.columns[categorical_feature_mask].tolist()

# Get list of non-categorical column names

non_categorical_columns = X.columns[~categorical_feature_mask].tolist()

# Apply numeric imputer

numeric_imputation_mapper = DataFrameMapper(
[([numeric_feature],Imputer(strategy="median")) input_df=True,
df_out=True
)
```

```
# Apply categorical imputer

 categorical_imputation_mapper = DataFrameMapper(
[(category_feature, CategoricalImputer()) input_df=True,
df_out=True
)
```

```
age          9
bp          12
sg          47
al          46
su          49
rbc        152
pc          65
pcc          4
ba           4
bgr         44
bu          19
sc          17
sod         87
pot         88
hemo        52
pcv         71
wc         106
rc         131
htn          2
dm           2
cad          2
appet        1
pe           1
ane          1
dtype: int64
```

[ ]:

```
# Combine the numeric and categorical transformations

numeric_categorical_union = FeatureUnion([
("num_mapper", numeric_imputation_mapper),
("cat_mapper", categorical_imputation_mapper)
])
```

[ ]:

```
# make a transformer: Dictifier
def dictifier(arr:'Array'):

"""This function is used to turn an array to a dictionary"""
# turn array to dataframe
dataframe = pd.DataFrame(arr)

# turn dataframe to a dictionary
df_dict = dataframe.to_dict('records')

# return results
return df_dict

# make a transformer
Dictifier = FunctionTransformer(dictifier, validate=False)
```

[ ]:

```
# Create full pipeline

pipeline = Pipeline([
("featureunion", numeric_categorical_union),
('dictifier', Dictifier),
("vectorizer", DictVectorizer(sort=False)),
("clf", xgb.XGBClassifier(max_depth=3))
])

# Perform cross-validation
cross_val_scores = cross_val_score(pipeline, X, y, scoring="roc_auc", cv=3)

# Print avg. AUC
print("3-fold AUC: ", np.mean(cross_val_scores))
```

```
3-fold AUC:  0.998637406769937
```

```
[ ]: # Fit Model
      # train test split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
     random_state=21, stratify=y)
```

```
[ ]:  # fit the model
     pipeline.fit(X_train, y_train)
```

```
Out[95]: Pipeline(memory=None,
              steps=[('featureunion', FeatureUnion(n_jobs=1,
                transformer_list=[('num_mapper', DataFrameMapper(default=False, df_out=True,
                 features=[(['age'], Imputer(axis=0, copy=True, missing_values='NaN', strategy='
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                silent=True, subsample=1))])
```

```
[ ]: Pipeline(memory=None,
     steps=[('featureunion', FeatureUnion(n_jobs=1,
     transformer_list=[('num_mapper', DataFrameMapper(default=False, df_out=True,
     features=[(['age'], Imputer(axis=0, copy=True, missing_values='NaN', strategy='reg_alpha=0,
                                  reg_lambda=1, scale_pos_weight=1, seed=None,
     silent=True, subsample=1))])
```

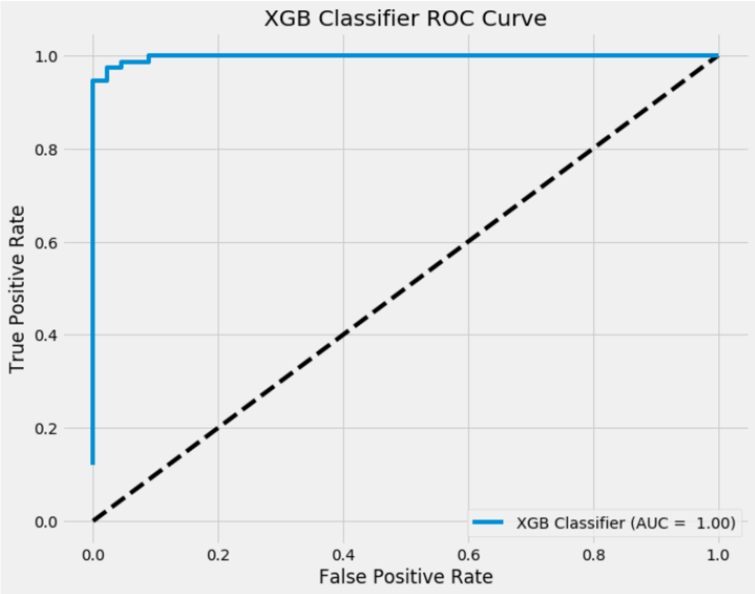
```
[ ]: ROC AUC Curve
       # predict on the test set
     y_pred_prob = pipeline.predict_proba(X_test)[:, 1] # [:, 1]:
```

```
[ ]:  # instantiate a roc_auc object
     ROC = roc_auc(y_test, y_pred_prob, model='XGB')
```

```
[ ]:  # AUC score
     ROC.auc()
```

```
Out[98]: 0.9976296296296295
```

```
[ ]:  # set figsize
     figsize(10,8)

     # plot stypple
     plt.style.use('fivethirtyeight')

     # plot roc
     ROC.plot_roc()
```




```
[ ]: # Feature Importance
      # set figsize

     figsize(10,8)

     # plot styple
     plt.style.use('fivethirtyeight')

     # plot feature importance
     xgb.plot_importance(pipeline.named_steps['clf'])
```

Out[101]: <matplotlib.axes._subplots.AxesSubplot at 0x1953acce438>